Lab #6

CSCI 4061 - Fall 2022 - 10/17/2022

Lab Preparation:

(1) Download lab files (git clone, git pull, or download from Canvas)

```
$ git clone https://github.umn.edu/csci-4061-fall-22/posted_labs.git
$ git pull
```

(2) Extract Lab Files from tar file

```
$ tar -xzvf lab_06_code.tar.gz
```

Lab Topics

- Midterm #1 review
- Pipe overview
- Review of all system calls
- Time for questions

pipe

Name: **pipe** - create a unidirectional pipe

Protype: `int pipe(int pipefd[2])`

Parameters: `int pipefd[2]` - 2D array to place FD's of opened pipe

Returns: 0 → On Success

-1 → On Failure

Helpful Shell Commands

Recommended to know and understand these commands

```
$ ls -l          → list directory
$ chmod          → change permissions
$ cd             → change directory
$ rm             → remove
$ mv             → move
$ mkdir          → make a directory
$ cp             → copy
$ pwd            → current working directory
$ link           → create a hard link
$ link -s        → create a symbolic link
…
and more         → know all the basics!
```

fork

Name: **fork** - create a child process

Protype:

```
pid_t fork(void);
```

Parameters: None

Returns: -1 → If error

0 → If child process

>0 → If parent process

wait

Name: **wait** - wait for <u>any</u> child process to change state

Protype:

```
pid_t wait(int *wstatus)
```

Parameters: `int *wstatus` - Int address to put return code of proc

Returns: -1 → If failure

PID → Process ID of child if success

HINT: For every fork there must be a wait

waitpid

Name: **waitpid** - wait for a specific process to change state

Protype:

```
pid_t waitpid(pid_t pid, int *wstatus, int options)
```

Parameters:

`pid_t pid` - Process ID to wait for

`int *wstatus` - Int addr to put return code of proc

`int options` - Wait options to use (see man page)

Returns:

-1 → If failure

0 → If WNOHANG used and child process exists

PID → Process ID of child if success

execl

First argument is always the name of the executable
e.g. execl("./hello", "hello", (char*)0);

Name: **execl** - execute a file

Protype:

```
int execl(const char *path, const char *arg0, ... /*, (char *)0 */);
```

Parameters:

`const char *path` - Path to executable file to run

`const char *arg0` - String that is the first argument

`...` - Any other strings to pass in as args

`(char *)0` - NULL char to signify end of args

Returns: -1 → If failure

Does not return on success

HINT: You should never return from exec, only on failure

execv

Name: **execv** - execute a file

Protype:

```
int execv(const char *path, char *const argv[])
```

Parameters:

`const char *path` - Path to executable file to run

`char *const argv[]` - Array of strings to pass as argos

Returns: -1 → If failure

Does not return on success

other forms of exec

Many other flavors of exec exist, check out the man page for more details

EXEC MAN PAGE

kill

Name: **kill** - send signal to process

Protype:

```
int kill(pid_t pid, int sig);
```

Parameters:

`pid_t pid` - Process ID to send signal to

`int sig` - Signal to send to process

Returns:

-1 → If error

0 → On success

Helpful Signals (signal.h):
SIGKILL → Kill process
SIGINT → Interrupt process

dup

Name: **dup** - duplicate a file descriptor

Protype:

```
int dup(int oldfd)
```

Parameters: `int oldfd` - File descriptor to duplicate

Returns: -1 → If error

fd → On success, returns the new FD value

Helpful File Descriptors (unistd.h):

STDIN_FILENO → 0
STDOUT_FILENO → 1
STDERR_FILENO → 2

dup2

Name: **dup2** - duplicate a file descriptor

Protype:

```
int dup2(int oldfd, int newfd)
```

Parameters:

```
int oldfd   - Old File Descriptor

int newfd   - New File Descriptor
```

newfd is adjusted so that it now refers to same open file descriptor as oldfd

Returns:

```
-1      → If error

newfd   → On success, returns the new FD value
```

creat

Name: **creat** - create a new file or rewrite an existing one

Protype:

```
int creat(const char *path, mode_t mode)
```

Parameters:

`const char *path` - File path to create

`mode_t mode` - File permissions when creating

Returns:

-1 → If error

fd → On success, returns the FD value

Helpful mode (fcntl.h):

```
S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH
```

r/w permissions for user, group and other

open - create a file

Name: **open** - open a file with create permissions

Protype:

```
int open(const char *pathname, int flags, mode_t mode)
```

Parameters:

`const char *pathname` - File path to open/create

`int flags` - Open flags to use

`mode_t mode` - File permissions to create with

Returns:

-1 → If error

fd → On success, returns the FD value

Helpful flags (fcntl.h):

`O_WRONLY|O_CREAT|O_TRUNC`

Write only, create flag and truncate flag

open

Name: **open** - open a file

Protype:

```
int open(const char *pathname, int flags)
```

Parameters:

```
const char *pathname    - File path to open
int flags               - Flags to open with
```

Returns:

```
-1      → If error
fd      → On success, returns the FD value
```

For flag values, visit man page: (man open)

close

Name: **close** - close an open file descriptor

Protype: `int close(int fd)`

Parameters: `int fd` - File Descriptor to close

Returns: -1 → If error

0 → On success, returns the FD value

read

Name: **read** - read from a file descriptor

Protype:

```
ssize_t read(int fd, void *buf, size_t count)
```

Parameters:

```
int fd          - File Descriptor to read from
void *buf       - Address to read into memory
size_t count    - Number of bytes to read
```

Returns:

```
-1          → If error
< count     → If fewer than count bytes in file
count       → Typical success when read all count byte
```

write

Name: **write** - write to a file descriptor

Protype:

```
ssize_t write(int fd, const void *buf, size_t count)
```

Parameters:

`int fd`	- File Descriptor to write to
`const void *buf`	- Memory address to write from
`size_t count`	- Number of bytes to write to fd

Returns:

-1	→ If error
count	→ On success, ***count*** bytes were written

fopen

Name: **fopen** - open a file stream

Protype:

```
FILE *fopen(const char *restrict pathname, const char *restrict mode)
```

Parameters:

`const char *restrict pathname` - File to open

`const char *restrict mode` - Mode to open file with

Returns:

NULL → If error

file stream → File stream opened on success

Open Modes:
"r" → read | "r+" → read/write | "w" → write & file create|
"w+" → read/write & file create | "a" → append
"a+" → read & append

fclose

Name: **fclose** - close a file stream

Protype:

```
int fclose(FILE *stream)
```

Parameters: `FILE *stream` - File stream to close

Returns:

EOF → If error

0 → On success

Read Size = size * nmemb

fread

Name: **fread** - read from a file stream

Protype:

```
size_t fread(void *restrict ptr, size_t size, size_t nmemb, FILE *restrict stream);
```

Parameters:

`void *restrict ptr` - Memory address to read **to**

`size_t size` - Size of each **nmemb** item

`size_t nmemb` - Number of items of **size** bytes to read

`FILE *restrict stream` - File stream to read **from**

Returns:

EOF → If error or end of file

bytes read → When read was successful

Write Size = size * nmemb

fwrite

Name: **fwrite** - write to a file stream

Protype:

```
size_t fwrite(void *restrict ptr, size_t size, size_t nmemb, FILE *restrict stream);
```

Parameters:

`void *restrict ptr` - Memory address to write **from**

`size_t size` - Size of each **nmemb** item

`size_t nmemb` - Number of items of **size** bytes to write

`FILE *restrict stream` - File stream to write **to**

Returns:

EOF → If error or end of file

bytes read → When read was successful

ftell

Name: **ftell** - get the current file position indicator

Protype: `long ftell(FILE *stream)`

Parameters: `FILE *stream` - Stream to get the FP indicator from

Returns: 0 → On success

-1 → On failure

fseek

Name: **fseek** - sets the file position indicated for a stream

Protype:

```
int fseek(FILE *stream, long offset, int whence)
```

Parameters:

`FILE *stream`	- Stream to seek in
`long offset`	- Number of bytes to seek from **whence**
`int whence`	- Where to seek from

<u>whence options:</u>
SEEK_SET → from the start of the file
SEEK_CUR → from the current file pointer position
SEEK_END → from the end of the file

Returns:

0	→ On success
-1	→ On failure

mkdir

Name: **mkdir** - make directories

Protype:

```
int mkdir(const char *pathname, mode_t mode)
```

Parameters: `const char *pathname` - Path to dir to make

`mode_t mode` - Permissions to make dir with

Returns: 0 → On success

-1 → On failure

opendir

Name: **opendir** - open a directory

Protype:

```
DIR *opendir(const char *name)
```

Parameters: `const char *name` - Directory name/path to open

Returns:

DIR pointer	→ On success
NULL	→ On failure

closdir

Name: **opendir** - open a directory

Protype: `int closedir(DIR *dirp)`

Parameters: `DIR *dirp` - Open directory pointer to close

Returns: 0 → On success

-1 → On failure

You need to call readdir for reach item in a directory

readdir

Name: **opendir** - open a directory

Protype: `struct dirent *readdir(DIR *dirp)`

Parameters: `DIR *dirp` - Open directory pointer

Returns: pointer to dirent structure → On success

NULL → On failure or end of dir

Checkout a dirent structure here: (readdir man page)

getcwd

Name: **getcwd** - get current working directory

Protype:

```
char *getcwd(char *buf, size_t size)
```

Parameters:

`char *buf` - mem location to place the cwd path in

`size_t size` - Max length of cwd path

Returns:

pointer to buffer → On success & buf passed in

pointer to an alloc str → On success & buf is NULL

NULL → On failure or end of dir

link

Name: **link** - make a new name for a file

Protype:

```
int link(const char *oldpath, const char *newpath)
```

Parameters:

```
const char *oldpath    - file to link
const char *newpath- new file name
```

Returns:

0 → On success

-1 → On failure

If this is the last link to the file, it will be deleted

unlink

Name: **unlink** - delete a name and possible the file it refers to

Protype:

```
int unlink(const char *pathname)
```

Parameters: `const char *pathname` - file to unlink

Returns:

0 → On success

-1 → On failure

symlink

Name: **symlink** - create a symbolic link file

Protype:

```
int symlink(const char *target, const char *linkpath)
```

Parameters: `const char *target` - target to link to

`const char *linkpath` - path to the symbolic link file to make

Returns: 0 → On success

-1 → On failure

stat

```
The statbuf pointer must have allocated memory:
struct stat *tmp = malloc(sizeof(struct stat));
struct stat tmp;   //Use &tmp
```

Name: **stat** - delete a name and possible the file it refers to

Protype:

```
int stat(const char *restrict pathname, struct stat *restrict statbuf)
```

Parameters:

```
const char *restrict pathname
```

- file path to stat

```
struct stat *restrict statbuf
```

- pointer to stat struct for results

Returns:

0 → On success

-1 → On failure